Lecture 7

Interactive graphics application

Interaction

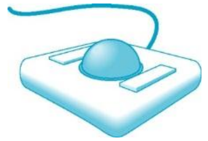
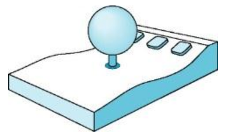
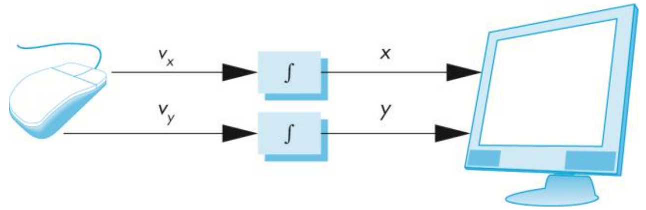
- An interactive graphics application is the one in which there is an interaction between the application and the user(the user acts…The application responds)
- The OpenGL graphics library core contains no interaction API
- To define an efficient good interactive graphics application, we must employ the undelaying specific window system
- We can provide basic interaction using a common simple interaction library: GLUT

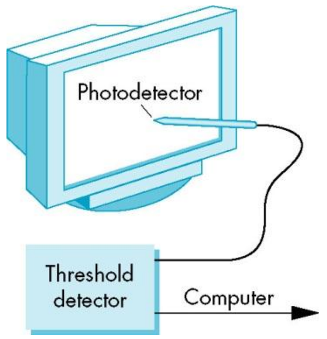
Input devices

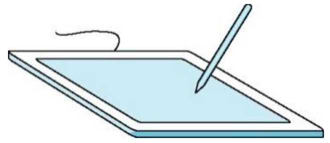
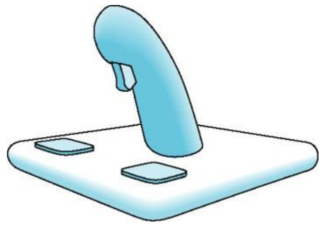
- Physical input devices
  - Keyboard, mouse, data tablet, trackball, joystick
- Logical input: API/widgets the application uses
  - Ex. Scanf(&x); printf(“%d”,x); get a location on the screen
  - These are the input devices from the application point of view
  - Scanf might get its input from the keyboard, file, another application through redirection
  - The application get a location from a mouse, trackball, data tablet or joystick using the same input API

Physical input devices

- Character generation devices
  - Keyboard, character recognition system, virtual keyboard, etc.
- Location input device with some buttons to initiate actions
  - Two degrees of freedom
    - Mouse (no zero position, no absolute displacement)
    - data tablet (we work in absolute positioning
    - Joysticks(has zero position and give mechanical feedback)
  - Six degrees of freedom
    - Space ball (up-down, front –back, left-right, three independent twist
  - More: data gloves, virtual reality sensors, position and orientation in 3D



Logical devices

- From the application program point of view, input is logically abstracted into logical devices or API that could be classified by:
  - What type of data they provide
  - The manner by which input device provide data to the application programs
    - Measure process (request/sampling/event)
    - Trigger(Enter on keyboard, button on a mouse, etc.)

Logical device continued

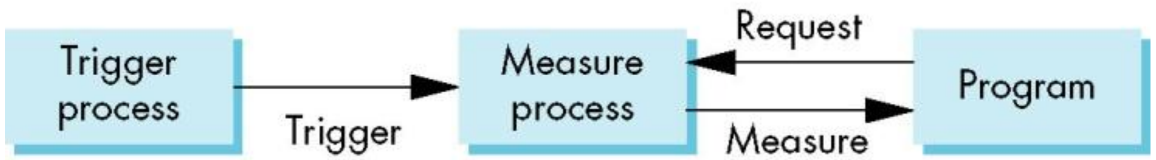
- Logical input devices could be classified according to the data they provide as:
  - Strings: characters (ex. Logical keyboard, Text Input devices)
  - Locator: position in world coordinates (ex. logical mouse)
    - An application may need to covert from window to world coor.
    - In windows API you get MouseDown and MouseMove events for stylus (this is the logical locator named mouse)
  - Pick: identify object on the screen
  - Choice: Identify item in a list of items (List box, Menus,…)
  - Valuators: provides analog input (Numerical input widget)
  - Stroke: provides array of location (software abstraction)

Logical device continued

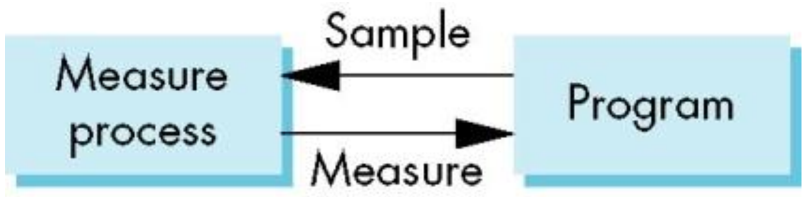
- Input mode (Relationship between measure process and trigger)
- Request mode (The application drive)
  - The application request (ex. scanf())
  - Measuring starts and continues until the trigger (Enter)
  - The measure is handed to the application
- Sampling mode (the application drive)
  - Measuring is continuous but a sample is handed to the program when needed (mouse position)
- Event mode (The input device drive)
  - When the device is triggered it fires an event with its own ID and the measure value at the triggering instant

Logical device continued: Input modes

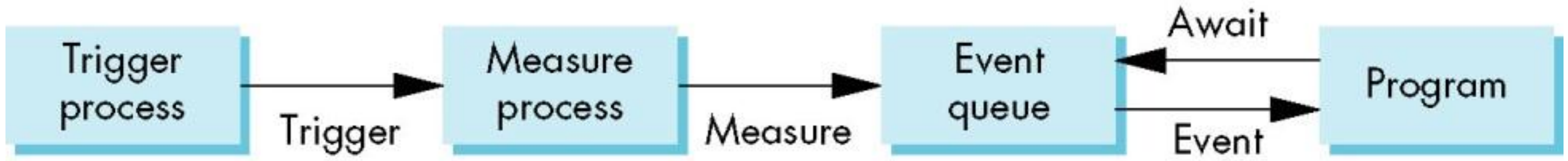
Request mode



Sample mode



Event mode



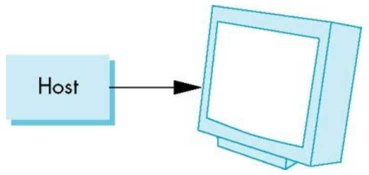
Client-Server and graphic application

Usually , The graphics application is not an isolated monolithic program that has its dedicated physical and logical input devices.

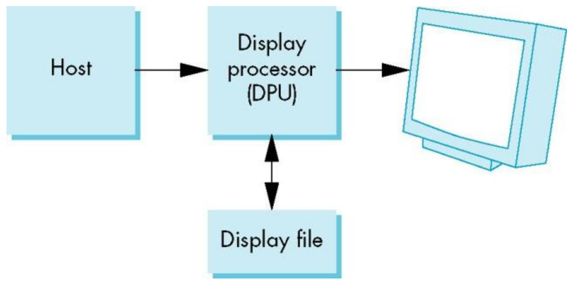
- It may run on a workstation that has other running programs that share input devices
- It may run on a workstation and is controlled (or co-controlled) from another workstation on a network
- One way to handle this complexity in a computing environment is the Client-Server relationships
  - In the computing environment there are servers the provide services and clients that consume these services
  - An entity may be client/server in the same time
  - The input services provided by a keyboard or a mouse on a workstation could be available to any clients on the network

Display lists and rendering modes

Original architecture of graphics systems: The host is responsible to refresh the display at a reasonable rate to prevent flicker which is impossible except for very few application



The solution is to use display processor that is responsible to refresh the screen. Nowadays display processors contains the graphics pipeline implemented on it. So they are called GPU



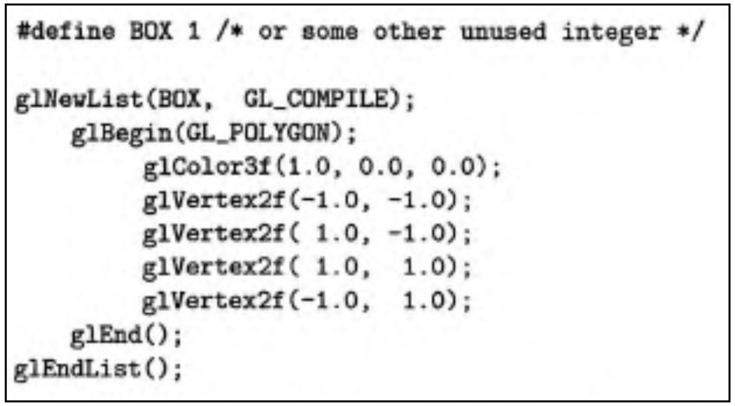
There are two rendering modes to use with the graphics processor:
Immediate mode: primitives defined in the application (main processor) , transferred immediately to the GPU (main processor) then rendered immediately (GPU)
Retained mode: The object is defined once (main processor), transmitted once to the GPU (main processor) stored on in GPU memory in a display list, rendered any time after by calling the list when needed

Definitions and execution of display list

Notes:

- The current state is applied on the display list commands
- The Display list can change the sate. This change is permanent until overwritten any time any where after
- To change and restore the state safely for a display list: Push the attributes and matrices at the beginning of the list, change the state as you wish inside the list, pop the attributes and the matrices at the end of the list

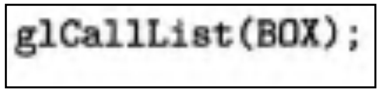
Definition of a display list
We can use GL_COMPILE to just create and store the list or GL_COMPILE_AND_EXECUTE to render it as well



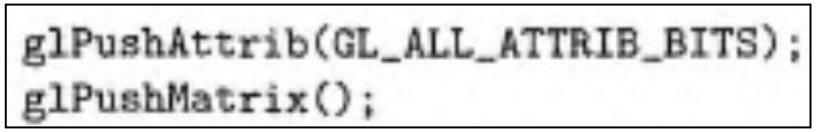
```
#define BOX 1 /* or some other unused integer */

glNewList(BOX,  GL_COMPILE);
    glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(-1.0, -1.0);
        glVertex2f( 1.0, -1.0);
        glVertex2f( 1.0,  1.0);
        glVertex2f(-1.0,  1.0);
    glEnd();
glEndList();
```

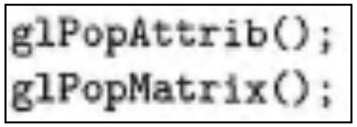
Calling of a display list, we can also call a set of consecutive lists in one call



```
glCallList(BOX);
```



```
glPushAttrib(GL_ALL_ATTRIB_BITS);
glPushMatrix();
```



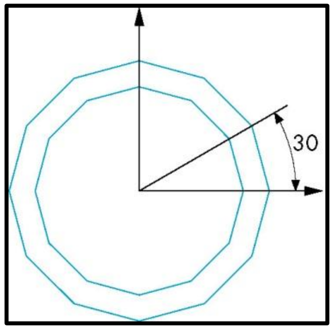
```
glPopAttrib();
glPopMatrix();
```

Text and display lists

- Displaying text using display list is more efficient than immediate mode rendering
  - When using raster text instead of transferring a block of points each time we output a specific character, we create a display list the draw that character and call the list each time we want display it
  - When using stroke text, we specify the primitives for each character once in a display list. This list is called each time the character is to be displayed

Text and display lists: Example creating a font

```
base = glGenLists(256); /* return index of first of 256
                           consecutive available ids */
for(i=0; i<256; i++)
{
        glNewList(base + i, GL_COMPILE);
        OurFont(i);
        glEndList();
}
```



```
void OurFont(char c)
    {
          switch(c)
          {
               case 'a':
                    ...
               break;
               case 'A':
                    ...
               break;
                    ...
          }
    }
```

```
case 'O':
    glTranslatef(0.5, 0.5, 0.0); /* move to center */
    glBegin{GL_QUAD_STRIP);
    for (i=0; i<=12; i++)  /* 12 vertices */
    {
          angle = 3.14159 /6.0 * i; /* 30 degrees in radians */
          glVertex2f(0.4*cos(angle)+0.5; 0.4*sin(angle)+0.5);
          glVertex2f(0.5*cos(angle)+0.5, 0.5*sin(angle)+0.5);
    }
    glEnd();
    break;
```

```
glListBase(base);
```

Set the base to give only the offset of the list

```
char *text_string;

glCallLists( (GLint) strlen(text_string), GL_BYTE, text_string);
```

Fonts in the GLUT

There are some ready to use fonts defined in the GLUT library. They are not defined using display lists

Example: using a GLUT stroke font

```
glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN, int character)
```

- The size of the character box is approximately 120 units, this units in not related to the unit you use in your program. So scaling for the character is often, required
- You start writing by a translation to the left-bottom of the first character
- The function does a translation to the bottom right of the character box to prepare for the next character
- Warning: these transformation affect the following graphics output. So it's a good idea to push the attributes and matrices before writing stroke text and pop them after

Example: using a GLUT raster font

```
glutBitmapCharacter(GLUT_BITMAP_8_BY_13, int character)
```

- Raster character is drawn directly in the buffer
- The raster position is controlled by glRasterPos()
- The function automatically prepare the raster position for the next character
- Raster text doe not affect the current sate of the transformation